Contents

Chapter 1

Deploying a High Throughput Computing Cluster

JIM BASNEY AND MIRON LIVNY

Department of Computer Sciences
University of Wisconsin-Madison
Wisconsin, USA
Email: *jbasney@cs.wisc.edu, miron@cs.wisc.edu*

1.1 Introduction

A High Throughput Computing (HTC) environment strives to provide large amounts of processing capacity to customers over long periods of time by exploiting existing computing resources on the network. To maximize processing capacity, the HTC environment must utilize heterogeneous resources. This requires a portable solution, which includes a resource management framework that effectively encapsulates the differences between resources in the cluster. To provide capacity over long periods of time, the environment must be reliable and maintainable—surviving software and hardware failures, allowing resources to join and leave the cluster easily, and enabling system upgrade and reconfiguration without significant downtime.

Most importantly, the system must meet the needs of resource owners, customers, and system administrators, since without the support of any one of these groups the HTC environment will fail. Resource owners donate the use of their resources to the customers of the HTC environment. Before they are willing to do this, the owners must be satisfied that their rights will be respected and the policies they specify will be enforced. Customers will use the HTC environment to run their applications only if the benefit of additional processing capacity is not outweighed by the cost of learning the complexities of the HTC system. System administrators will install and maintain the system only if it provides a tangible benefit to its users which outweighs the cost of maintaining the system.

Resources in the HTC cluster may be distributively owned, meaning that the control over powerful computing resources in the cluster is distributed among many individuals and small groups. For example, many individuals in an organization may each have "ownership" of a powerful desktop workstation. The willingness to share a resource with the HTC environment may vary for each resource owner. The cluster may include some resources which are dedicated to HTC, others which are unavailable for HTC during certain hours or when the resource is otherwise in use, and still others which are available to only specific HTC customers and applications. Even when resources are available for HTC, the application may be allowed only limited access to the components of the resource and may be preempted at any time. Additionally, distributed ownership often results in decentralized maintenance, when resource owners maintain and configure each resource for a specific use. This adds an additional degree of resource heterogeneity to the cluster.

The deployment of an HTC cluster is both a technological and sociological process. The HTC software must be robust and feature-rich to meet the needs of resource owners, customers, and system administrators. However, even the best HTC system must have support within an organization before it can be deployed effectively. Often, developing this support is an evolutionary process. First, an HTC "evangelist" deploys a small cluster with his or her own resources and with resources donated by HTC "allies." The HTC evangelist then helps a few HTC customers effectively use the small cluster. By demonstrating the benefits of the HTC cluster to these customers, the evangelist creates demand for HTC within the organization. At this point, the customers may approach the system administrators and policy makers to request that the pool be expanded, or the customers may approach resource owners directly to ask for additional resource donations. The customers are also in the position to help more of their colleagues become customers of the cluster.

This chapter describes some of the challenges faced by software developers and system administrators when deploying an HTC cluster, and some of the approaches for meeting those challenges based on the experience of the developers and administrators of the Condor HTC environment, which has been deployed for over a decade at the University of Wisconsin-Madison Computer Sciences department [1]. We focus on those issues which become more important when the cluster grows large and is maintained for many years. In our experience, it is not exotic scheduling algorithms and mechanisms which make an HTC environment successful, but an emphasis on usability, flexibility, reliability, and maintainability.

1.2 Condor Overview

While a detailed description of Condor is outside the scope of this chapter, we give a short overview here to provide a concrete example of an HTC system architecture.

In Condor, each customer is represented by a customer agent, which manages a queue of application descriptions and sends resource requests to the matchmaker. Each resource is represented by a resource agent, which implements the policies of the resource owner and sends resource offers to the matchmaker. The matchmaker is responsible for finding matches between resource requests and resource offers and

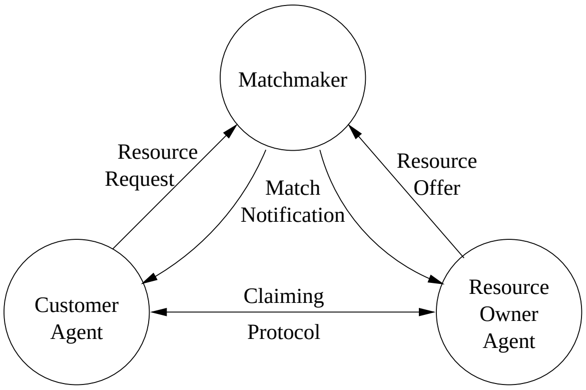

Figure 1.1 Condor resource management architecture.

notifying the agents when a match is found. Upon notification, the customer agent and the resource agent perform a claiming protocol to initiate the allocation. This architecture is illustrated in Figure 1.1.

Resource requests and offers contain constraints which specify if a match is acceptable. So, the customer agent includes a constraint in its resource request which specifies which resource offers are acceptable. For example, the customer agent may desire only resources running a specific operating system. The resource agent includes a constraint in the resource offer which specifies which requests it will service. For example, the resource agent may only be willing to service requests made by a specific customer. An offer matches a request when both constraints are satisfied.

The matchmaker implements systemwide policies by imposing its own set of constraints on matches. For example, the matchmaker implements a customer priority mechanism by matching resource requests in priority order, so resource requests from customers with better priorities have a better opportunity to find a match. The matchmaker may preempt allocations by matching a resource with a new request to maintain a fair distribution of allocations. The customer agent or the resource agent may also choose to break the allocation at any time.

1.3 Software Development

The developer of an HTC system must overcome four primary challenges: utilization of heterogeneous resources, evolution of network protocols, remote file access, and utilization of nondedicated resources. The utilization of heterogeneous resources requires system portability, which can be obtained most effectively through a layered system design. A smooth evolution of network protocols is required for a system where resources and customer needs are constantly changing, requiring the deployment of new features in the HTC system. A remote file access mechanism

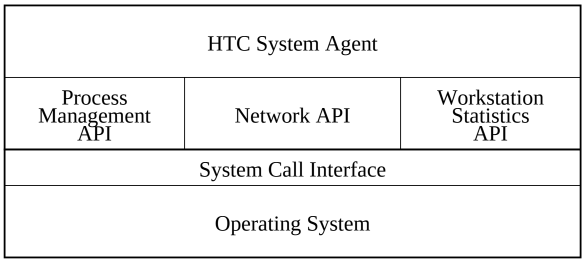

Figure 1.2 Layered software architecture.

guarantees that an application will be able to access its data files from any workstation in the cluster. Finally, the utilization of non-dedicated resources requires the ability to preempt and resume an application using a checkpoint mechanism.

1.3.1 Layered Software Architecture

The HTC system is a client of the workstation operating system. In particular, the HTC system relies on the host operating system to provide network communication, process management, and workstation statistics. Since the interface to these services differs on each operating system, the portability of the HTC system will benefit from a layered software architecture, as shown in Figure 1.2. The system is written to a system independent API, reducing the cost of porting to a new architecture, because the nonportable code is isolated in the API libraries.

The network API provides both connection-oriented and connectionless, reliable and unreliable interfaces, with many mechanisms for authentication and encryption. It performs all conversions between host and network integer byte order automatically, checks for overflows (when, for example, sending an integer from a 64-bit workstation to a 32-bit workstation), and provides standard host address lookup mechanisms.

The process management API provides the ability to create, suspend, unsuspend, and kill a process to enable the HTC system to control the execution of a customer's application. A parent process may pass state to new child processes, including network connections, open files, environment variables, and security attributes. Since the customer's application does not necessarily use any HTC libraries, the API implementation must not assume that the child process also runs an instance of the same API.

The workstation statistics API reports the information necessary to implement the resource owner policies and verify that the customer application requirements are met. The resource owner policy may refer to the CPU, network, and disk load, the time of day, the time since the last keyboard or mouse activity, the amount of available swap space, and other resource attributes. The customer application may, for example, require a specific operating system and CPU architecture, and a minimum amount of available physical memory, disk space, and network bandwidth.

Many libraries already exist to provide portable system services to applications. For example, the XDR (eXternal Data Representation) library provides translation services between data representations on different operating systems and processor architectures. There are obvious benefits to using standard libraries when developing the HTC system. Since the libraries are already written, the HTC developers save time by using the libraries in lieu of developing new libraries. The developers save time when porting the HTC system to new platforms if the existing libraries are already available for the new platform. Also, one can assume that a library which is already in wide use is better tested than a new library. However, these benefits are not always realized. General purpose libraries are often poor fits to the specific needs of an HTC environment, and so using such libraries adds unnecessary baggage to the system. Additionally, the library may not be available for all platforms or may work incorrectly on some platforms, resulting in porting and debugging work for the HTC developer.

1.3.2 Layered Resource Management Architecture

The resource management architecture of the HTC environment also benefits from a layered system design framework. Figure 1.3 shows such an architecture used in Condor. This approach yields a modular system design, where the interface to each system layer is defined in the resource management model, allowing the implementation of each layer to change so long as the interface continues to be supported. Customized customer agents may be developed with different scheduling algorithms optimized for specific classes of applications. Resource owner agents may be customized to implement desired access control mechanisms. The matchmaker (part of the System Layer) may be upgraded to utilize new resource management algorithms without requiring an upgrade of other agents in the cluster.

This architecture separates the advertising, matchmaking, and claiming protocols. The agents advertise resource offers and requests asynchronously to the matchmaker, and the matchmaker notifies the agents when a match is found. Since the matchmaker is not involved in the claiming protocol, the protocol may be customized for specific types of agents, and may be modified without affecting the negotiator. The matchmaker does not need to know the details of allocation establishment, and so many different allocation protocols may be easily supported by the same matchmaker [2].

1.3.3 Protocol Flexibility

As the distributed system evolves to provide new and improved services, the network protocols will be affected. Often, the protocols are augmented to transfer additional information. This often requires that all components of the distributed system be updated to recognize the additional information. In a large HTC system, it is often inconvenient to update all components at one time, and so new features are not deployed until a future major system upgrade.

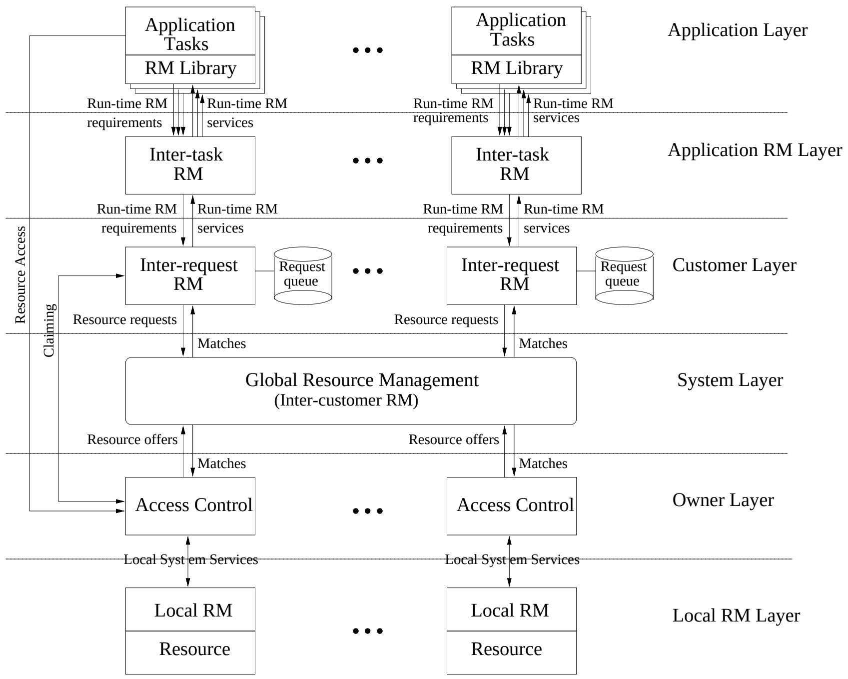

Figure 1.3 Layered resource management architecture.

To support this evolution, the HTC network protocols may utilize a general-purpose data format which allows more flexibility. For example, Figure 1.4 illustrates the protocol data format used throughout Condor. A leading integer specifies the protocol action to be performed, and the named parameter list which follows specifies the data associated with that protocol action. This is similar to an RPC protocol, where an integer first specifies the RPC being invoked and the parameters of the call follow. The parameters in the list[1] are named, so the receiver may iterate through the parameter list, or may simply look up the values for the named parameters of interest. To enhance the protocol, new parameters are simply added to the parameter list. Backward compatibility is ensured, since older agents will ignore the new parameters and new agents are written to accept packets with or without the new parameters.

1.3.4 Remote File Access

A remote file access mechanism guarantees that an HTC application will have access to its data files from any workstation in the cluster. This mechanism may use an

[1]The named parameter list is a use of the Condor ClassAd resource management language, which is described in more detail in [3].

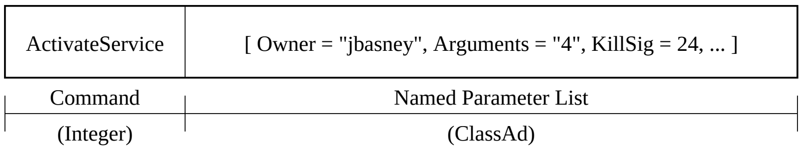

Figure 1.4 Example of protocol data format.

existing distributed file system, it may stage data files on the workstation's local disk, or it may redirect file I/O system calls to a remote file server via system call interposition.

To effectively use an existing distributed file system, the HTC environment must authenticate the customer's application to that file system. For example, NFS authenticates via user ID, while AFS and NTFS authenticate via Kerberos and NT server credentials. To run the customer's application with the appropriate distributed file system rights, the HTC environment may require administrator privileges on the remote workstation, the ability to transparently forward credentials, or the ability to obtain the customer's credentials (for example, using the customer's password). Alternatively, the customer may be required to grant file access permission to the HTC system before submitting the application for execution.

To implement data file staging, the HTC system requires a list of input files from the customer for each application. The system then transfers these input files to the local disk of the remote workstation before running the application. The system is responsible for gathering up the application's output files and transferring them to a destination specified by the customer when the application has completed. This requires free disk space on the remote workstation and the bandwidth to transfer the data files at the start and end of each allocation. For large data files, this results in high start-up and tear-down costs compared to a block file access mechanism provided by a distributed file system or redirected file I/O system calls.

To redirect file I/O system calls, the HTC environment must interpose itself between the application and the operating system and service file I/O system calls itself [4], [5], as illustrated in Figure 1.5. This may be accomplished by linking the application with an interposition library or by trapping system calls through an operating system interface. The HTC environment invokes an RPC to perform the file operation on a server with access to the customer's data files. Since the file operations are performed at the system call level, this may result in many high latency operations, reducing the performance of the application. Read-ahead and write-behind caching can effectively reduce this latency.

Redirecting file I/O system calls has the significant benefit that it places no file system requirement on the remote workstation. This enables the HTC environment to utilize a greater number of resources. The drawback is that developing and maintaining a portable interposition system can be very difficult, since different operating systems provide different interposition techniques and the system call

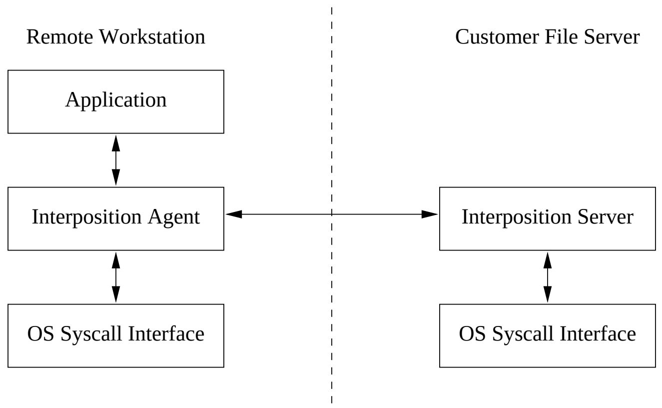

Figure 1.5 System call interposition.

interface differs on each operating system and often changes with each new operating system release. If an interposition library is used, then multiple compilers and linking techniques (static versus dynamic linking, 32-bit versus 64-bit executables, etc.) must often be supported. Thus, supporting the redirection of file I/O system calls can be a significant portion of the cost of deploying an HTC cluster.

1.3.5 Checkpointing

A checkpoint of an executing program is a snapshot of its state which can be used to restart the program from that state at a later time. Computing systems have traditionally employed checkpointing to provide reliability: when a compute node fails, the program running on that node can be restarted from its most recent checkpoint, either on that same node once it is restored or potentially on another available node. Checkpointing also enables preemptive-resume scheduling. All parties involved in an allocation can break the allocation at any time without losing the work already accomplished by simply checkpointing the application. Thus, a long running application can make progress even when allocations last for relatively short periods of time. Due to the opportunistic nature of non-dedicated resources in a cluster environment, any attempt to deliver HTC has to rely on a checkpointing mechanism.

Since most workstation operating systems do not provide kernel-level checkpointing services, an HTC environment must often rely on user-level checkpointing. Developing a portable, robust user-level checkpointing mechanism is a significant challenge for the developer of an HTC environment [6], [7], since operating system APIs for querying and setting process state vary and are often incomplete.

1.4 System Administration

The administrator of an HTC environment answers to resource owners, customers, and policy makers. It is the administrator's responsibility to guarantee that the HTC environment enforces the access policies of resource owners. Since resources in a cluster are often heterogeneous and distributively owned, these policies are often complicated and vary from resource to resource. The administrator is also responsible for ensuring that customers receive valuable service from the HTC environment. This involves working with customers to understand the needs of their applications and developing an approach for running each application in the HTC environment. It also often requires detecting application failures, investigating the causes of the failures, and developing solutions to avoid future failures. Finally, the system administrator often must demonstrate to policy makers that the HTC environment is meeting stated goals. This requires accounting of system usage and availability.

1.4.1 Access Policies

The resource access policy specifies who may use a resource, how they may use it, and when they may use it. The administrator determines an access policy in consultation with the resource owner and implements that policy through a configuration of the HTC environment. The configuration mechanism must be rich enough to express a wide variety of access policies.

One method for policy specification is to define a set of expressions which specify when an application may begin using a resource and when and how an application must stop using a resource. For example, consider the following set of expressions:

- The *Requirements* expression evaluates to *true* when an application may begin using the resource.

- The *Rank* expression evaluates to a larger numerical value for applications which the owner would prefer over others.

- The *Suspend* expression evaluates to *true* when the active application should be immediately suspended.

- The *Continue* expression evaluates to *true* when the active application should be immediately unsuspended.

- The *Vacate* expression evaluates to *true* when the active application should be notified to stop using the resource. The application may continue using the resource for a short time to save its intermediate results.

- The *Kill* expression evaluates to *true* when the active application should be immediately stopped.

These expressions may refer to both application attributes (such as the identity of the customer) and resource attributes (such as the time since the last keyboard event).

Consider the following example access policy. The owner of a desktop workstation will allow an application to use the workstation only when the owner has not been using it for fifteen minutes (i.e., the keyboard and mouse have been idle for that period of time, and the CPU load is low). Furthermore, the owner may prefer to run applications owned by "jbasney@cs.wisc.edu" over other applications. Finally, when the owner returns (i.e., the keyboard and mouse are no longer idle), the application should be immediately suspended. If the owner continues to use the workstation for five minutes, the application should be notified to stop using the resource. The application should be immediately stopped if it is still running five minutes later. This policy is implemented with the following expressions:

Requirements = (KeyboardIdle > 15∗Minute) && (LoadAvg < 0.3)
Rank = (Customer == "jbasney@cs.wisc.edu") ? 1 : 0
Suspend = (KeyboardIdle < Minute)
Continue = (KeyboardIdle > 2∗Minute)
Vacate = (SuspendTime > 5∗Minute)
Kill = (VacateTime > 5∗Minute)

Suspend, Vacate, and Kill provide three mechanisms for the owner to preempt a running application. Each mechanism results in different costs for the application and the resource owner. Suspend keeps the application state on the resource (i.e., in virtual memory) but suspends execution. This benefits the application when the owner reclaims the resource for only a short time, because it allows the application to immediately resume its execution (via Continue) when the resource is available again. Vacate allows the application to save any immediate results (i.e., checkpoint) before relinquishing the resource. Alternatively, Kill does not allow the application to save intermediate results, so the application's work is lost. Thus, by enabling Suspend and/or Vacate, the resource owner allows the application to better utilize the resource at the cost of a prolonged preemption.

The cost of application placement and preemption are significant factors in setting good access policies. Application placement requires transferring the executable, checkpoint, and data files to the remote host, and preemption requires transferring checkpoint and data files to a new remote host or to storage. Since the application checkpoint contains the memory state of the process, it may be very large (100MB+), and therefore transferring this data over the network may require a large amount of bandwidth. These costs motivate the use of Suspend, which avoids the cost of preemption and placement when the resource is reclaimed for only a short period of time.

In the example policy above, the application is allowed five minutes after the Vacate event to save its state before the Kill event occurs. On low-bandwidth networks, a large application will not be able to complete its checkpoint in this time period. To improve throughput in such an environment, the administrator will want to attempt to negotiate a longer Vacate interval with the resource owners.

In cases when the chance of successful checkpoint is very low, the administrator can configure the workstation to not attempt a Vacate, since it will only cause unnecessary network traffic. For example:

Vacate = (SuspendTime > 5∗Minute) && (JobImageSize < 100∗MB)
Kill = (JobImageSize < 100∗MB) ? (VacateTime > 5∗Minute) :
    (SuspendTime > 5∗Minute)

The administrator may use a periodic checkpoint mechanism to reduce the amount of work lost as a result of failed preemption checkpoints and other system failures (network failure, workstation hardware failure, etc.). Applications are configured to perform checkpoints periodically so that they can rollback to the most recent checkpoint in the case of a failure. Since performing periodic checkpoints consumes CPU, network, and disk resources, the administrator must balance the periodic checkpoint frequency with the expected rate of failure.

The administrator may also steer matchmaking to utilize resources efficiently when network bandwidth is limited. Strategies include steering applications with greater network requirements to resources with greater available network bandwidth and longer expected allocation times. The administrator uses his or her knowledge of the networking infrastructure, network load, and application requirements to effectively configure the HTC environment.

1.4.2 Reliability

Reliability is a primary concern in an HTC environment because of the variety of risks of failure and the special needs of HTC customers. The HTC environment relies on the services provided by the network, hardware, and operating system of each node on the network. The system must strive to mask failures in these components and recover gracefully. The HTC environment must also handle failures in components of the HTC system itself, as such software failures are to be expected in such a complicated distributed system. HTC customers rely on the environment to manage the execution of their applications. Since these applications may have long execution times (weeks, months, or years), it is essential that applications survive these failures.

A distributed file system can be a frequent cause of system failures. The HTC environment may rely on a distributed file system to provide file access to applications, system configuration files, executables, and log files. A file system failure may appear to a long running as a failed system call. Many applications will simply abort when this occurs. The HTC environment can put this application "on hold" until the file system recovers and then restart the application from a previous checkpoint. If the HTC executables are accessed via a distributed file system, a file system failure may cause process crashes due to page faults which can not be serviced.[2] The HTC software must also react appropriately when configuration files

[2]This is caused by the operating system using the executable file as the backing store for the text pages of the process. When the executable file is inaccessible, the process fails once a page fault occurs in the text segment.

and log files are temporarily inaccessible.

Since the HTC system is responsible for enforcing the policies of the resource owner, it is essential that the system processes don't fail and leave running applications unattended. The HTC environment can provide functionality which enables the administrator to enhance the reliability of these processes. A Master process can be dedicated to monitoring the other HTC processes on a workstation to detect failures and invoke recovery mechanisms. This process can also cache executable files on the local disk of the workstation to avoid unserviceable page faults. Additionally, this process may serve as an administrative module of the HTC environment, to report which services are currently running, allow the administrator to start and stop services, and to detect and react to configuration changes and system upgrades. In a large cluster, a Master process can dramatically reduce the cost of system reconfiguration and upgrade by automatically retrieving new files from the distributed file system and gracefully restarting local services to take advantage of the upgrade.

There are a number of complications which arise when implementing a mechanism to allow applications to recover from system failures. The HTC software must be able to detect the difference between normal application termination, abnormal termination due to an environmental failure, and abnormal termination due to an unrecoverable application error. One approach is to monitor the system calls performed by the application to detect the source of failures. Alternatively, the system could defer to the customer, asking the customer to alert the system when an application terminates abnormally. The HTC software must also choose the correct checkpoint to use for restart. It is possible for an environmental failure to cause a failure in the application which results in an abnormal termination after a significant delay. During that delay, the application may have performed a checkpoint. The HTC system should ideally rollback to a checkpoint which was performed before the failure occurred. Finally, the HTC software must decide when it is safe to restart the application. If the source of the failure is known, the system could poll to determine when the failure has been resolved. Alternatively, the system could contact the customer or system administrator and request a response when the failure has been diagnosed and resolved.

An HTC environment is particularly susceptible to the "problem of one bad node." This problem occurs when one node in the cluster enters into a state which causes application failures (the node may run out of swap space, a memory module may go bad, network file services may fail, etc.). Thus, whenever an application begins running on this node, it terminates abnormally. The HTC environment must avoid naively running application after application on this workstation, or this single node will be able to quickly drain the system of applications (or put them all "on hold"). Thus, for an application failure, the system must determine if the application failed due to the specific environment of the current node, due to the current environment of the entire cluster, or due to an application error. This could be determined heuristically: if the application fails consistently on different nodes, then it is reasonable to conclude that the entire cluster environment is experiencing the problem for this application; if different applications fail on the same node, then it is reasonable to conclude that the particular node is to blame and should be disabled.

To summarize, the HTC environment must be prepared for failures and must automate failure recovery for common failures. This need grows significantly as the cluster grows in size. By successfully handling common failures, the HTC environment frees the administrator to investigate less common failures and to otherwise concentrate on managing the system for efficiency.

1.4.3 Problem Diagnosis via System Logs

Failures will occur in even the most reliable HTC environment—applications may terminate abnormally, resource owners may report that their resource is being used inappropriately, customers may report that they are not receiving a fair amount of service, etc. System log files are the primary tools for diagnosing system failures. Using good log files, the administrator should be able to reconstruct the events leading up to the failure, which in most cases will uncover the cause of the failure. Log files are also essential in determining if a failure actually occurred. For example, the resource owner who reports that the resource policy has been breached may be mistaken or may have a misunderstanding of the policy implementation. Knowing what occurred on the resource helps the administrator to decide if the customer's policy should be modified or if there is a system problem. Maintenance of good system logs requires the decision of what information to log and a mechanism for writing and accessing system logs.

Table 1.1 lists some of the useful information which can be logged by the HTC system. The information is categorized by HTC subsystem to show the importance of effective log file organization. For example, when investigating a reported scheduling problem, the administrator will first focus only on the scheduling logs, avoiding the distraction of unrelated log messages. There are a number of potentially useful organizations or views for system logs. For example, when investigating the failure of a specific application, it may be useful to trace the life of the application through the different subsystems to see when the application was scheduled, how long the allocation lasted, which system calls were performed by the application during the allocation, and which resource policy action (if any) coincided with the application failure. This example argues for an application-specific view of the system logs. Customer or resource specific views are also helpful. The administrator also needs to be able to view different levels of logging detail when diagnosing a problem. These views may be implemented by logging each view to a separate file or by tagging each log entry with a descriptive key which specifies the views to which it belongs.

System logs can grow to an unbounded size, so it is necessary to manage the amount of historical log information which is kept by the system. The logging facility can be configurable, so that detailed logs are kept for an administrator-specified period of time, and then only summaries are kept for older information. For example, when the administrator arrives on Monday morning to discover a

Table 1.1 HTC Environment Logs

application log:	system call trace checkpoint information and statistics remote I/O trace with statistics errors occurring during the allocation
customer log:	allocation information and statistics application arrival and termination matchmaking and claiming errors
resource log:	allocation information and statistics policy action trace
master log:	HTC agent (re-)starts administrative commands agent upgrades
scheduling log:	record of all matches allocation history (accounting)
security log:	record of all rejected requests record of all authenticated actions

problem report, it is useful to have detailed logs from the weekend to diagnose the problem. It is also useful to have a historical summary of system usage which goes back many years, to track changes in cluster capacity and customer demand, so the administrator may report the return received on the investment in the HTC system.

Managing distributed log files can be cumbersome, often requiring the administrator to remotely access workstation after workstation to follow the migration of an application or to examine many instances of a particular problem. One alternative is to store logs centrally on a file server or a customized log server. Another alternative is to provide a single interface to the distributed log files by installing logging agents on each workstation which will respond to log queries made by a client application.

1.4.4 Monitoring and Accounting

In addition to diagnostic logs, the HTC environment provides system monitoring and accounting facilities to the administrator. This allows the administrator to assess the current and historical state of the system and to track system usage.

For example, Figure 1.6 shows a stacked graph of the number of allocated ("Condor"), available ("Idle"), and unavailable ("Owner") resources in the UW-Madison Computer Sciences Condor cluster for the month of September 1998. From this visualization, the administrator can conclude that:

- Approximately 100 resources were added to the cluster during the month.
- Over 50 percent of the cluster capacity was harnessed by HTC applications

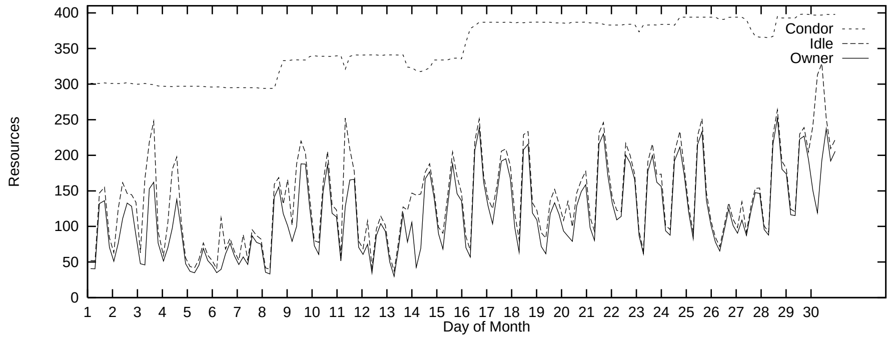

Figure 1.6 Monitoring resource usage.

during the month.

- Resource availability followed a daily cyclic pattern, where more resources were available for HTC during the night.
- On average, more resources were available to HTC applications on weekends compared to weekdays.
- Very few resources were left unutilized by either the owner or an HTC application, except on a few occasions (for example, September 14 and 30). A large number of unutilized resources is a sign of system inefficiency or a shortage of customer requests.

Figure 1.7 shows a stacked graph of the number of idle and running applications in the Condor cluster during September 1998. The daily cyclic pattern of available resources is seen again here in the number of running applications. Also, a shortage of customer requests is shown to be the cause of the unutilized resources on September 14.

Using the same accounting facilities which generated these graphs, the administrator can see that HTC applications were allocated approximately 155 thousand CPU hours during the month.[3]

1.4.5 Security

An HTC environment is potentially vulnerable to both resource and customer attacks. A resource attack occurs when an unauthorized user gains access to a resource via the HTC environment or when an authorized user violates the resource owner's access policy. A customer attack occurs when the customer's account or data files are compromised via the HTC environment.

[3]These statistics are available online at http://www.cs.wisc.edu/condor/.

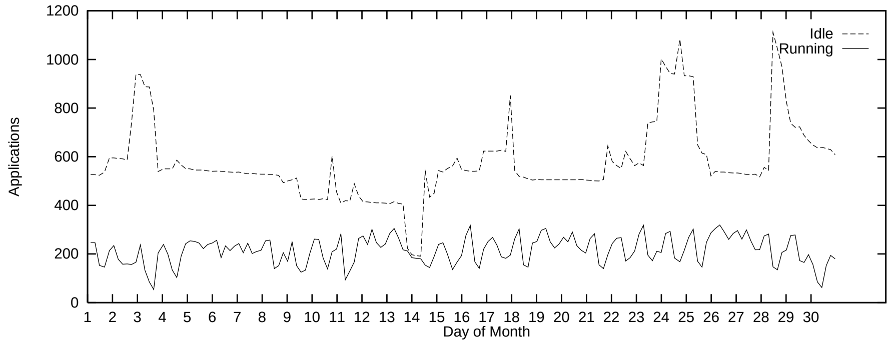

Figure 1.7 Monitoring HTC applications.

Protecting the resource from unauthorized access requires an effective user authentication mechanism. The resource owner may explicitly list authorized users in the access policy, using the Requirements expression. For example:

Requirements = (Customer == "jbasney@cs.wisc.edu" ||
Customer == "miron@cs.wisc.edu")

The HTC system must ensure the validity of the Customer attribute. The resource agent can verify the Customer attribute by requesting that the customer agent digitally sign the resource request with the customer's private key. The resource agent will then verify the signature and the fact that the *Requirements* expression is *true* for this resource request. Alternatively, the resource agent may establish a trust relationship with the customer agent, and rely on the customer agent to set the Customer attribute appropriately.

Protecting against violations of the resource owner's access policy requires that the resource agent maintain control over the application and monitor its activity. The resource agent may use the operating system API to set resource consumption limits for the application. It may also run the application under a "guest" account which provides only limited access to the workstation. To limit file system access, the agent may use operating system APIs to set the file system root directory to a "sandbox" directory. Perhaps the most effective approach, however, is to intercept the system calls performed by the application via the operating system interposition interface. This allows the resource agent to monitor all system access performed by the application and to enforce the owner's access policy by aborting any system calls which would violate it.

To submit an application to the HTC environment, the customer must grant the system access to the application executable and data files. This may be done by transferring the files to a directory or file system dedicated for HTC applications. In this case, the application may run with credentials specific to the HTC environment, without the need for the customer's credentials. When the application terminates, the customer retrieves the output files from the dedicated area. In this case, the customer's executable and data files are potentially vulnerable to snooping or modification. Alternatively, the HTC environment may run the application with the customer's file system credentials, allowing the customer's files to be accessed directly and conveniently. In this case, the customer's account is potentially vulnerable.

An untrustworthy resource agent can potentially mount a customer attack. To allow remote execution, the application must have access to its data files via a remote file access mechanism. The resource agent, therefore, has the opportunity to manipulate or replace the application to steal the customer's data or modify the customer's files. In the case when the application runs with the customer's file system credentials, the attacker has the opportunity to access all of the customer's files and install a trojan horse in the customer's file system. To protect against this attack, the HTC environment must ensure that all resource agents are trustworthy. Resource agents may be authenticated cryptographically or the cluster can be restricted to include only resource agents on trusted hosts (authenticated via IP-address,[4] for example).

Unencrypted network streams provide another potential vulnerability. Customer data and file system credentials sent unencrypted over the network are vulnerable to snooping. Unencrypted streams are also potentially vulnerable to hijacking, which would allow an attacker to modify executable and data files and gain unauthorized system access.

Finally, as with any network-enabled agent, HTC agents are potentially vulnerable to the common buffer-overflow attack. HTC developers and administrators should be aware of this potential attack and should assure themselves (using software quality assurance techniques) that the HTC system implementation is not vulnerable.

1.4.6 Remote Customers

Traditionally, customers were granted access to cluster computing environments via an account on one or more workstations in the cluster. The customer would transfer application data files to this workstation and compile the application for the cluster environment. However, it can be more convenient for both customers and administrators to provide remote access to the HTC cluster instead. The customer installs a customer agent on his or her workstation, and the administrator allows that agent access to the HTC cluster. The customer is no longer required to manually transfer data files, since an HTC remote file access mechanism is available from the customer's workstation. The administrator is no longer required to create a workstation account for the customer in the cluster, but instead must only create an HTC account.

Remote customers may require special consideration when configuring the HTC environment. These customer agents may not be considered as trustworthy as local customer agents, and so additional security precautions may be required. Addition-

[4] potentially vulnerable to IP-spoofing attacks

ally, these customer agents may connect to the cluster over a wide area network, which provides limited bandwidth, decreased reliability, and additional security concerns. Thus, there is a greater need for caching in the remote file access mechanism, local storage of intermediate files (including checkpoints) in the cluster, and encrypted network communication. The administrator may have limited access to the remote customer agents, since the agent runs on a remote workstation, so agent configuration and log file access may require assistance from the customer unless the HTC environment provides administration access mechanisms or the customer grants the administrator access to the remote workstation.

1.5 Summary

Deploying an HTC cluster presents many challenges for the developers and administrators of the HTC environment. The HTC software must be portable, reliable, and maintainable. A layered architecture with flexible network protocols provides such a framework. Remote file access and checkpointing mechanisms allow the HTC environment to utilize distributively owned, non-dedicated resources, but these mechanisms carry significant development and maintenance costs. The HTC system administrator must effectively balance the needs of resource owners and HTC customers, using an expressive policy configuration language. The HTC software must provide reliable, secure services with effective logging and accounting tools for monitoring resource usage and diagnosing problems. At its best, the HTC environment provides convenient access to cluster resources which are otherwise inaccessible, due to heterogeneity, distributed ownership, and other complexities. The HTC challenge is in effectively managing these complexities for the HTC customers, resource owners, and administrators.

1.6 Bibliography

[1] M. Litzkow and M. Livny. Experience with the Condor Distributed Batch System. *IEEE Workshop on Experimental Distributed Systems*, Huntsville, Alabama, October 1990.

[2] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, Illinois, July 1998.

[3] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for High Throughput Computing. *SPEEDUP Journal*, vol. 11(1), pages 36–40, June 1997.

[4] M. Litzkow. Remote UNIX - Turning Idle Workstations into Cycle Servers *Proceedings of the 1987 Usenix Summer Conference*, Phoenix, Arizona, 1987.

[5] M. Jones. Interposition Agents: Transparently Interposing User Code at the System Interface. *14th ACM Symposium on Operating Principles*, vol. 27(5), December 1993.

[6] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System. *University of Wisconsin-Madison Computer Sciences Technical Report 1346*, April 1997.

[7] J. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent Checkpointing under Unix. *Conference Proceedings, Usenix Winter 1995 Technical Conference*, New Orleans, Louisiana, pages 213–223, January 1995.

Index